

Lehrstuhl für
Rechnerarchitektur & Parallele Systeme
Prof. Dr. Martin Schulz
Dominic Prinz
Jakob Schäffeler

Lehrstuhl für
Design Automation
Prof. Dr.-Ing. Robert Wille
Stefan Engels

Einführung in die Rechnerarchitektur

Wintersemester 2024/2025

Übungsblatt 2: RISC-V Assembly

28.10.2024 – 01.11.2024

1 RISC-V Simulator Einrichtung

In ERA wird zur Simulation eines RISC-V Prozessors **QtRvSim** – ein Projekt der Tschechischen Technischen Universität – verwendet. Für die erste Einrichtung folgen Sie bitte den folgenden Schritten:

1. Laden Sie sich die passende Installationsdatei für Ihr Betriebssystem herunter: <https://github.com/cvut/qtrvsim/releases/tag/v0.9.8>
 - Ubuntu-User können auch folgendes PPA verwenden: `ppa:qtrvsimteam/ppa`
 - Windows-User benutzen die Datei mit `mingw32` im Namenoder verwenden Sie die Web-Version (experimentell): <https://comparch.edu.cvut.cz/qtrvsim/app>
2. Belassen Sie die Einstellungen wie sie sind: „No pipeline no cache“ und klicken Sie auf „Example“.
3. In der oberen Hälfte sehen Sie die Register inkl. der zugehörigen Mnemonics und Werte.
4. Links sehen Sie die auszuführenden Instruktionen. Die Instruktionen eines Programms beginnen wie im RISC-V Ökosystem üblich bei Adresse `0x200`.
5. Sie können mithilfe der Reiter „Core“ und „template.S“ zwischen der Prozessor- und Source Code-Ansicht wechseln.
6. Klicken Sie auf „Compile Source and update memory“ (blauer Pfeil nach unten) um den aktuell ausgewählten Source Code zu kompilieren und zu laden.
7. Anschließend können Sie das Programm mit dem Play-Button starten. Als Beispiel wird der Text „Hello world.“ rechts auf dem Terminal ausgegeben.

8. Weitere Beispiele finden Sie hier: <https://gitlab.fel.cvut.cz/b35apo/stud-support/-/tree/master/seminaries/qtrvsim>
9. Tipp: Probieren Sie sich hier aus!

2 Registerbenutzung und (symbolische) Konstanten

- a) Machen Sie sich nochmals mit den Registern und deren üblicher Verwendung vertraut. Die Register sind in Tabelle 1 abgebildet.
- b) Alle für uns relevanten Befehle sind in Tabelle 2 gelistet. Nutzen Sie möglichst wenige Befehle, um folgende Aufgaben zu bewältigen.
- Laden Sie in das Register `a0` den Wert 0 (sog. Nullen).

`addi a0, zero, 0`
`xorv a0, a0, a0`

- Laden Sie die Konstante `0xF0000000` in das Register `t1`.

`lui t1, 0xF0000`

- Laden Sie die Konstante `0xABAD1DEA` in das Register `a3`. Verwenden Sie keine Pseudobefehle.

$$0xABAD1DEA = 4 + 0xFFFFFDEA$$

$$4 = 0xABAD1DEA - (-0x00000216)$$

$$\begin{array}{r} 0xABAD1DEA \\ + \\ 0x00000216 \\ \hline 0xABAD2000 \end{array}$$

`lui a3, 0xABAD2` `0x216`
`addi a3, a3, -0x216`

`li a3, 0xABAD1DEA`

3 Einfache Befehle

a) Welchen Wert hat das Register a0 nach der Ausführung des folgenden Programms:

```
main:
    addi a0, zero, -534
    lui a0, 0
    srli a0, a0, 10
    add a0, a0, a0
```

a0
-534
0
0
0

b) Übersetzen Sie die folgenden Terme in RISC-V Assembler. Verwenden Sie dabei nur Befehle aus Tabelle 2.

- $a_0 := a_0 - a_1$

sub a0, a0, a1

- $a_0 := a_0 + 2^{10}$

addi t1, zero, 1
slli t1, t1, 10
add a0, a0, t1

addi a0, a0, 1024

- $a_0 := a_0 + 2^{11}$

addi t1, zero, 1
slli t1, t1, 11
add a0, a0, t1

- $a_0 := a_0 \cdot 2 + 123$ (vorzeichenlose Multiplikation)

slli a0, a0, 1
addi a0, a0, 123

- $a_0 := a_0 \cdot 5$ (vorzeichenlose Multiplikation)

slli t0, a0, 2
add a0, a0, t0

4 Bitmasken und -hacks

Setzen Sie die folgenden Aufgaben um indem Sie Bitmasken und -hacks verwenden.

- a) $a_0 = \lfloor a_0/16 \rfloor$ (Ergebnis abgerundet, vorzeichenlose Division)

slli a0, a0, 4

$$256 = 1.0000.0000$$

$$2^8 - 1 = 1111.1111$$

- b) $a_0 = a_0 \bmod 256$ (vorzeichenlos)

andi a0, a0, 255

Modulo: Rest einer Integer-Division

z.B. $9 \bmod 4 = 1$, da $9 = 2 \cdot 4 + 1$

$0 \bmod 4 = 0$, da $0 = 0 \cdot 4 + 0$

$4 \bmod 4 = 0$, da $4 = 1 \cdot 4 + 0$

- c) Kopieren Sie die unteren 16 Bit von a1 in die unteren 16 Bit von a0 und die unteren 16 Bit von a2 in die oberen 16 Bit von a0.

slli a1, a1, 16

slli a1, a1, 16

slli a2, a2, 16

or a0, a1, a2

$$\begin{array}{r} a_1 \ 0 \dots 0 \ + \\ a_2 \ 1 \dots 1 \ 0 \end{array}$$

- d) Setzen Sie das a1-te Bit in a0 auf Eins. Die restlichen Bits sollen unverändert bleiben.

addi t0, zero, 1

sll t0, t0, a1

or a0, a0, t0

$$000 \overset{a_1}{1} 0000$$

- e) Bonus: Setzen Sie das a1-te Bit in a0 auf Null. Die restlichen Bits sollen unverändert bleiben.

addi -0, zero, 1

sll t0, t0, a1

or a0, a0, t0

xor a0, a0, t0

// Bit gerührt auf 1

// 1 xor 1 gerührt 0

5 Vorüberlegungen zur Hausaufgabe

- a) Was passiert, wenn man die in der Zentralübung erwähnte Formel zur Berechnung des Werts einer Binärzahl ($a = \sum_{i=0}^n a_i \cdot 2^i$) auf negative Indizes erweitert?

$$2^{-1} = \frac{1}{2} \quad 2^{-2} = \frac{1}{4} \quad \frac{1}{8} \quad \frac{1}{16} \quad \dots$$

- b) Welchen Wertebereich kann man mit 8 binären Vorkommastellen (ohne Vorzeichen) und 8 binären Nachkommastellen (8.8) erreichen?

Vorkomma: $2^8 - 1 = 255$

Nachkomma: $2^{-8} = \frac{1}{256} = 0,00039 \dots$

$1 - 2^{-8} = \frac{255}{256} = 0,996 \dots$ Wertebereich $[0,255,996]$

- c) Wieviele Bit bräuchte man mindestens, um Zahlen von 0 bis 100 darzustellen, sodass der Abstand zwischen zwei darstellbaren Werten maximal 0.005 beträgt?

0-1 2 1
4 2
⋮ ⋮

$$\lceil \log_2(100) \rceil = 7$$

$$\lceil -\log_2(0,005) \rceil = 8$$

- d) Wie sieht die Addition bzw. Subtraktion in Festkommarechnung auf einem Blatt Papier aus? Was muss man beachten?

- e) Wie sieht die Multiplikation in Festkommarechnung auf einem Blatt Papier aus? Was muss man beachten?

z.B. 8.8 VK NK

$$a = 1,5 = \frac{0000.0001}{1000.0000} \quad 1000.0000 = 384$$

$$b = 3,25 = \frac{0000.0011}{0100.0000} = 832$$

Nachkommastellen Eig = Summe Nachk. der Operanden

$384 \cdot 832 = 10000.0100 \quad 1110.0000 \quad 0000.0000$

4 879

f) Wie sieht die Zahl π im 88-Format aus? Was muss man beachten?

$$\lfloor 3,14 \cdot 2^8 \rfloor = 804$$

$$\frac{804}{2^8} = 3,14$$

g) Wie kann man beliebige Zahlen in Festkommazahlen umwandeln?

siehe f

6 Festkommarechnung (Hausaufgabe 02)

Bearbeitung und Abgabe der Hausaufgabe 02 auf <https://artemis.in.tum.de/courses/401> bis Sonntag, den 03.11.2024, 23:59 Uhr.

© 2024 Lehrstuhl für
Rechnerarchitektur &
Parallele Systeme
alle Rechte vorbehalten

7 Referenzmaterial

Name	Register Number	Use
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0-2	x5-7	Temporary registers
s0/fp	x8	Saved register/Frame pointer
s1	x9	Saved register
a0-1	x10-11	Function arguments/Return values
a2-7	x12-17	Function arguments
s2-11	x18-27	Saved registers
t3-6	x28-31	Temporary registers

Abbildung 1: RISC-V 32-Bit Register

op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	000	–	I	lb rd, imm(rs1)	load byte	rd = SignExt([Address] _{7:0})
0000011 (3)	001	–	I	lh rd, imm(rs1)	load half	rd = SignExt([Address] _{15:0})
0000011 (3)	010	–	I	lw rd, imm(rs1)	load word	rd = [Address] _{31:0}
0000011 (3)	100	–	I	lbu rd, imm(rs1)	load byte unsigned	rd = ZeroExt([Address] _{7:0})
0000011 (3)	101	–	I	lhu rd, imm(rs1)	load half unsigned	rd = ZeroExt([Address] _{15:0})
0010011 (19)	000	–	I	addi rd, rs1, imm	add immediate	rd = rs1 + SignExt(imm)
0010011 (19)	001	0000000*	I	slli rd, rs1, uimm	shift left logical immediate	rd = rs1 << uimm
0010011 (19)	010	–	I	slti rd, rs1, imm	set less than immediate	rd = (rs1 < SignExt(imm))
0010011 (19)	011	–	I	sltiu rd, rs1, imm	set less than imm. unsigned	rd = (rs1 < SignExt(imm))
0010011 (19)	100	–	I	xori rd, rs1, imm	xor immediate	rd = rs1 ^ SignExt(imm)
0010011 (19)	101	0000000*	I	srlr rd, rs1, uimm	shift right logical immediate	rd = rs1 >> uimm
0010011 (19)	101	0100000*	I	srair rd, rs1, uimm	shift right arithmetic imm.	rd = rs1 >>> uimm
0010011 (19)	110	–	I	ori rd, rs1, imm	or immediate	rd = rs1 SignExt(imm)
0010011 (19)	111	–	I	andi rd, rs1, imm	and immediate	rd = rs1 & SignExt(imm)
0010111 (23)	–	–	U	auipc rd, upimm	add upper immediate to PC	rd = {upimm, 12'b0} + PC
0100011 (35)	000	–	S	sb rs2, imm(rs1)	store byte	[Address] _{7:0} = rs2 _{7:0}
0100011 (35)	001	–	S	sh rs2, imm(rs1)	store half	[Address] _{15:0} = rs2 _{15:0}
0100011 (35)	010	–	S	sw rs2, imm(rs1)	store word	[Address] _{31:0} = rs2
0110011 (51)	000	0000000	R	add rd, rs1, rs2	add	rd = rs1 + rs2
0110011 (51)	000	0100000	R	sub rd, rs1, rs2	sub	rd = rs1 – rs2
0110011 (51)	001	0000000	R	sll rd, rs1, rs2	shift left logical	rd = rs1 << rs2 _{4:0}
0110011 (51)	010	0000000	R	slt rd, rs1, rs2	set less than	rd = (rs1 < rs2)
0110011 (51)	011	0000000	R	sltu rd, rs1, rs2	set less than unsigned	rd = (rs1 < rs2)
0110011 (51)	100	0000000	R	xor rd, rs1, rs2	xor	rd = rs1 ^ rs2
0110011 (51)	101	0000000	R	srl rd, rs1, rs2	shift right logical	rd = rs1 >> rs2 _{4:0}
0110011 (51)	101	0100000	R	sra rd, rs1, rs2	shift right arithmetic	rd = rs1 >>> rs2 _{4:0}
0110011 (51)	110	0000000	R	or rd, rs1, rs2	or	rd = rs1 rs2
0110011 (51)	111	0000000	R	and rd, rs1, rs2	and	rd = rs1 & rs2
0110111 (55)	–	–	U	lui rd, upimm	load upper immediate	rd = {upimm, 12'b0}
1100011 (99)	000	–	B	beq rs1, rs2, label	branch if =	if (rs1 == rs2) PC = BTA
1100011 (99)	001	–	B	bne rs1, rs2, label	branch if ≠	if (rs1 ≠ rs2) PC = BTA
1100011 (99)	100	–	B	blt rs1, rs2, label	branch if <	if (rs1 < rs2) PC = BTA
1100011 (99)	101	–	B	bge rs1, rs2, label	branch if ≥	if (rs1 ≥ rs2) PC = BTA
1100011 (99)	110	–	B	bltu rs1, rs2, label	branch if < unsigned	if (rs1 < rs2) PC = BTA
1100011 (99)	111	–	B	bgeu rs1, rs2, label	branch if ≥ unsigned	if (rs1 ≥ rs2) PC = BTA
1100111 (103)	000	–	I	jalr rd, rs1, imm	jump and link register	PC = rs1 + SignExt(imm), rd = PC + 4
1101111 (111)	–	–	J	jal rd, label	jump and link	PC = JTA, rd = PC + 4

* Encoded in instr_{31:25}, the upper seven bits of the immediate field

Abbildung 2: RISC-V 32-Bit Integerbefehle